ASSIGNMENT -4

NAME : Pradyumna Kumar Das UID:22BCS13175

1. Longest Nice Substring

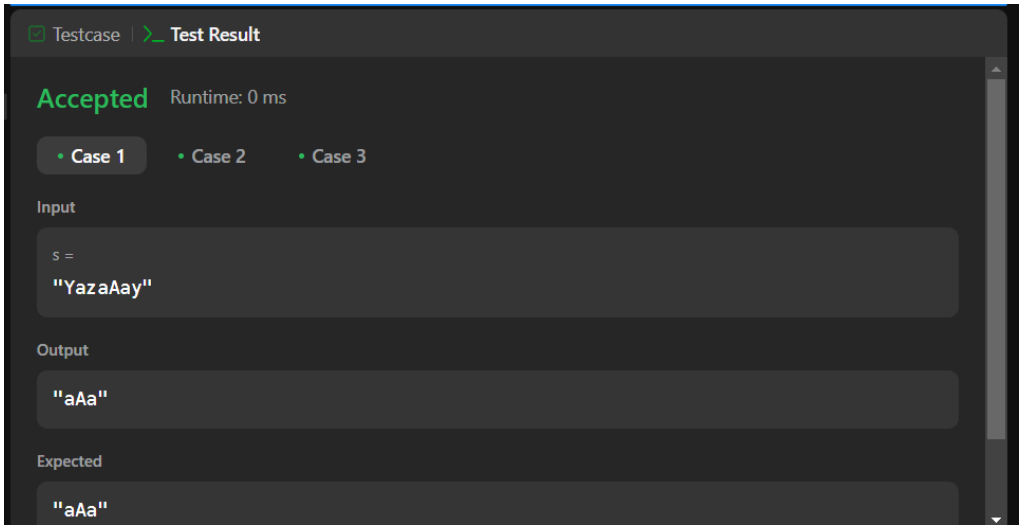
A string s is **nice** if, for every letter of the alphabet that s contains, it appears **both** in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Given a string s, return *the longest **substring** of s that is **nice**. If there are multiple, return the substring of the **earliest** occurrence. If there are none, return an empty string*.

SOLUTION:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2)
            return "";
        Set<Character> set = new HashSet<>();
        for (char ch : s.toCharArray())
            set.add(ch);
        for (int i=0; i<s.length(); i++) {
            char current = s.charAt(i);
            if (set.contains(Character.toUpperCase(current)) &&
    set.contains(Character.toLowerCase(current)))
                continue;
```

```
            String str1 = longestNiceSubstring(s.substring(0, i));
            String str2 = longestNiceSubstring(s.substring(i + 1));
            return str1.length() >= str2.length() ? str1 : str2;
        }
        return s;
    }
}
```



2. Reverse Bits

   Reverse bits of a given 32 bits unsigned integer.

   Note:

   Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.

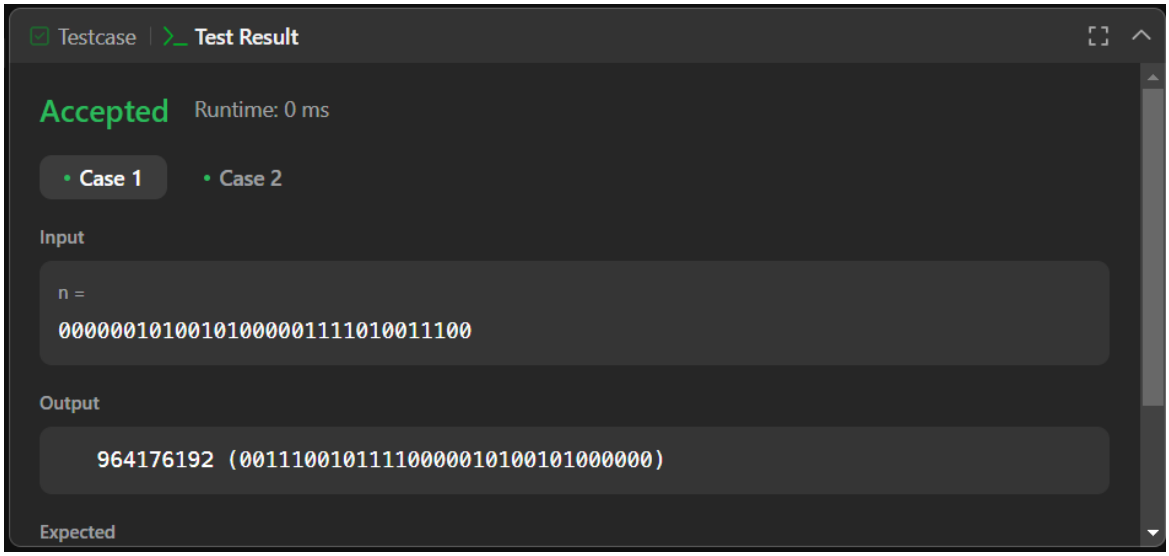
   In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in Example 2 above, the input represents the signed integer -3 and the output represents the signed integer

SOLUTION:

```
public class Solution {
    public int reverseBits(int n) {
        int result = 0;

        for (int i = 0; i < 32; i++) {
            result |= (n & 1) << (31 - i);
            n >>>= 1;
        }

        return result;
    }
}
```



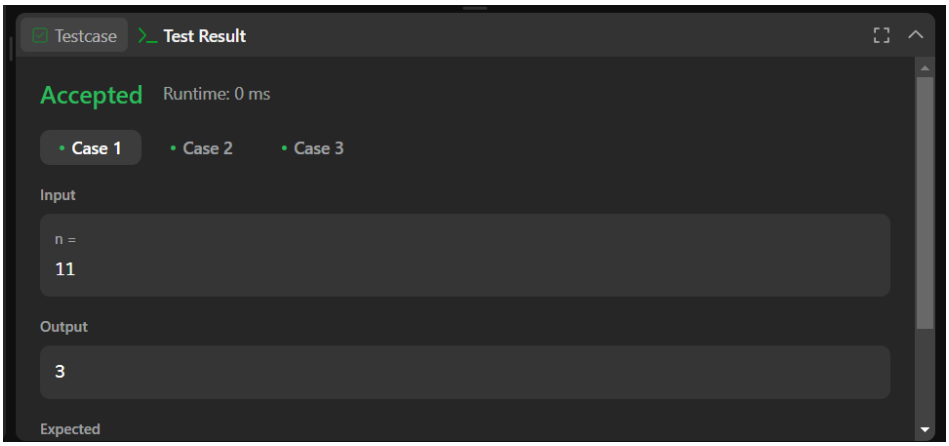
3. **Number of 1 Bits**

Given a positive integer n, write a function that returns the number of set

bits in its binary representation (also known as the Hamming weight).

SOLUTION:

```
public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```



4. **Maximum Subarray**

Companies

Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

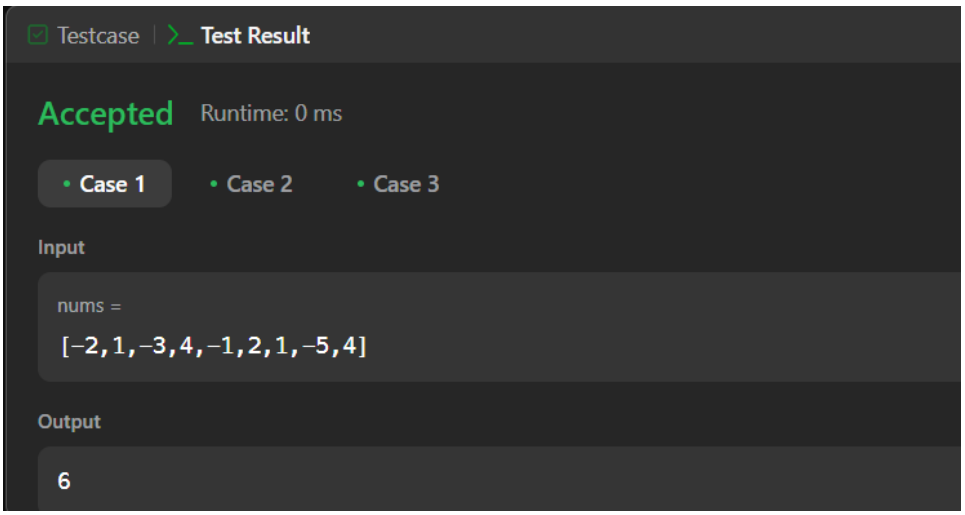
SOLUTION:

```
public class Solution {
```

```
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }
}
```



5. **Search a 2D Matrix II**

Write an efficient algorithm that searches for a value target in an m x n integer matrix . This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

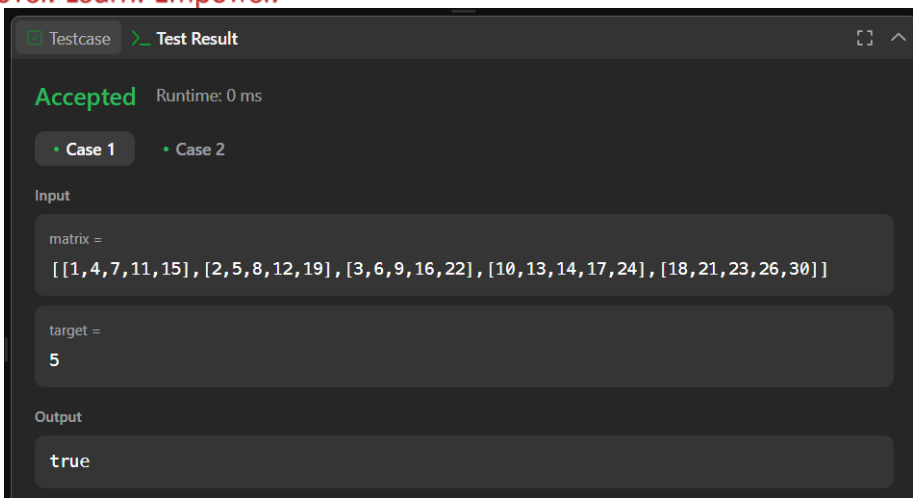
SOLUTION:

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {

        int n = matrix.length;
        int m = matrix[0].length;
        int row = 0 ;
        int col = m-1;
        while(row<n && col>=0)
        {
            if(matrix[row][col]==target)  return true ;
            else if (matrix[row][col]<target)  row++;
            else {
                col--;
            }
        }

        return false ;

    }
}
```



6. Super Pow

Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

SOLUTION:

```
class Solution {

    public int superPow(int a, int[] b) {
        if (a % 1337 == 0) return 0;
        int result = 1;
        for (int digit : b) {
            result = modPow(result, 10) * modPow(a, digit) % 1337;
        }
        return result;
    }

    private int modPow(int base, int exponent) {
        base %= 1337;
        int result = 1;
```

```
        for (int i = 0; i < exponent; i++) {
            result = (result * base) % 1337;
        }
        return result;
    }
}
```